IV. General Remarks Concerning This Response

Claims 1-50 are currently pending in the present application. Claims 1, 13, 24, and 33 have been amended to fix antecedent basis errors or typographical errors. No claims have been canceled herein.

Claim 8, which was missing in the original set of claims, has been added herein.

Claims 1 and 13, which incorrectly mentioned "the reception software" in the original claims (which was noted as lacking an antecedent basis by the Office action), have been corrected to refer to the antecedent term "the request implementation software" in both claims.

Applicant notes that the claims have not been amended to avoid prior art; all of the amendments to the claims were minor corrections that did not substantively change the subject matter in the claims. Reconsideration of the claims is respectfully requested.

Several errors in the specification concerning reference numbers were noted by the Office action; the specification has been corrected herein.

Errors in Figure 3 and Figure 4 that were noted by the Office action have been corrected. A set of formal drawings are being submitted by mail separately from this response, which is being faxed.

The Office action also noted that the abstract was too long; a new abstract is being submitted herein.

V. 35 U.S.C. § 102(e)–Anticipation–*Devine et al.*

The Office action has rejected claims 1-7 and 9-50 under 35 U.S.C. § 102(e) as anticipated by Davis, "Secure Customer Interface for Web Based Data Management", U.S. Patent No. 6,598,167 B2, filed 09/26/1997, issued 07/22/2003. This rejection is respectfully traversed.

The claims of the present patent application are directed to a particular software architecture comprising beans that respond to propagated events for implementing functionality related to digital certificates; each of the independent claims introduces a requirement of at least one bean. However, the claim rejections completely ignore the specific language in the claims that recite these software architectural features. For example, the second element of amended independent claim 1 reads as follows:

> at least one reception bean, communicatively coupled to the request implementation software and the distributed processing system, that generates an event object in response to receiving the request to generate a digital certificate from the distributed processing system;

The rejection of independent claim 1 states that this feature is found in Devine et al. at column 12, line 16, to column 13, line 25, and at column 7, line 20, through column 8, line 16, along with Figure 17 and its description in columns 15-16.

Applicant strongly disagrees that Devine et al. discloses the claimed features. Moreover, if Devine et al. disclosed each of the claimed features as argued in the rejections, then the rejection should have been written so that there was a clear correspondence between disclosed features and claimed features. However, the rejection confusingly points to several columns of text for most features without presenting any additional arguments that explain how a specific feature of Devine et al., e.g., as might be found within a few lines of text in Devine et al., discloses a specific feature of the claimed invention. By pointing to several columns of text, the rejection obfuscates the fact that Devine et al. does not disclose the claimed features. Applicant asserts that the rejection purposefully points to large amounts of text for a given claimed feature of the present invention in order to appear as having a correct format for an anticipation rejection while completely lacking substantive merit.

In order to emphasize that Devine et al. does not disclose the claimed features, Applicant copies hereinbelow the portions of Devine et al. that have been applied by the rejection against the second element of claim 1 along with other portions of Devine et al. that have been applied against other claims.

Column 7, line 20, to column 8, line 16:

> FIG. 3 illustrates an example client GUI presented to the client/customer as a browser web page 60 providing, for example, a suite 70 of network management applications, which may include: Traffic Monitor 72; an Alarm Monitor 73; a Network Manager 74 and Intelligent Routing 75. Access to network functionality is also provided through Report Requester 76, which provides the ability to define and request a variety of reports for the client/customer and a Message Center 77 for providing enhancements and functionality to traditional e-mail communications by providing access to user requested reports and bulk data. Additional network MCI Internet applications not illustrated in FIG. 3 include Online Invoice, relating to electronic invoicing and Service Inquiry related to Trouble Ticket Management.
>
> As shown in FIGS. 2 and 3, the browser resident GUI of the present invention implements a single object, COBackPlane which keeps track of all the client applications, and which has capabilities to start, stop, and provide references to any one of the client applications.
>
> The backplane 12 and the client applications use a browser 14 such as the Microsoft Explorer versions 4.0.1 or higher for an access and distribution mechanism. Although the backplane is initiated with a browser 14, the client applications are generally isolated from the browser in that they typically present their user interfaces in a separate frame, rather than sitting inside a Web page.
>
> The backplane architecture is implemented with several primary classes. These classes include COBackPlane, COApp, COAppImpl, COParm. and COAppFrame classes. COBackPlane 12 is an application backplane which launches the applications 54a, 54b, typically implemented as COApp. COBackPlane 12 is generally implemented as a Java applet and is launched by the Web browser 14. This backplane applet is responsible for launching and closing the COApps.
>
> When the backplane is implemented as an applet, it overrides standard Applet methods unit( ), start( ), stop( ) and run( ). In the unit( ) method, the backplane applet obtains a COUser user context object. The COUser object holds information such as user profile, applications and their entitlements. The user's configuration and application

entitlements provided in the COUser context are used to construct the application toolbar and Inbox applications. When an application toolbar icon is clicked, a particular COApp is launched by launchApp( ) method. The launched application then may use the backplane for inter-application communications, including retrieving Inbox data.

The COBackPlane 12 includes methods for providing a reference to a particular COApp, for interoperation. For example, the COBackPlane class provides a getApp( ) method which returns references to application objects by name. Once retrieved in this manner, the application object's public interface may be used directly.

The use of a set of common objects for implementing the various functions provided by the system of the present invention, and particularly the use of browser based objects to launch applications and pass data therebetween is more fully described in the above referenced copending application GRAPHICAL USER INTERFACE FOR WEB ENABLED APPLICATIONS, and Appendix A, attached to that application, provides descriptions for the common objects which includes various classes and interfaces with their properties and methods.

Column 12, line 16, through column 13, line 25:

Another communications issue involving the secure communications link, is the trust associated with allowing the download of the Java common objects used by the present invention, as discussed earlier with respect to the browser, since the Java objects used in the present invention require that the user authorize disk and I/O access by the Java object.

Digital Certificates, such as those developed by VeriSign, Inc. entitled Verisign Digital ID.TM. provide a means to simultaneously verify the server to the user, and to verify the source of the Java object to be downloaded as a trusted source as will hereinafter be described in greater detail.

As illustrated in FIG. 10, the process starts with the browser launch as indicated at step 280, and the entry of the enterprise URL, such as HTTPS://www.enterprise.com as indicated at step 282. Following a successful connection, the SSL handshake protocol is initiated as indicated at step 283. When a SSL client and server first start communicating, they agree on a protocol version, select cryptographic algorithms, authenticate the server (or optionally authenticate each other) and use public-key encryption techniques to generate shared secrets. These processes are performed in the handshake protocol, which can be summarized as follows: The client sends a client hello message to which

the server must respond with a server hello message, or else a fatal error will occur and the connection will fail. The client hello and server hello are used to establish security enhancement capabilities between client and server. The client hello and server hello establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random.

Following the hello messages, the server will send its digital certificate. Alternately, a server key exchange message may be sent, if it is required (e.g. if their server has no certificate, or if its certificate is for signing only). Once the server is authenticated, it may optionally request a certificate from the client, if that is appropriate to the cipher suite selected.

The server will then send the server hello done message, indicating that the hello-message phase of the handshake is complete. The server will then wait for a client response. If the server has sent a certificate request Message, the client must send either the certificate message or a no_certificate alert. The client key exchange message is now sent, and the content of that message will depend on the public key algorithm selected between the client hello and the server hello. If the client has sent a certificate with signing ability, a digitally-signed certificate verify message is sent to explicitly verify the certificate.

At this point, a change cipher spec message is sent by the client, and the client copies the pending Cipher Spec into the current Cipher Spec. The client then immediately sends the finished message under the new algorithms, keys, and secrets. In response, the server will send its own change cipher spec message, transfer the pending to the current Cipher Spec, and send its finished message under the new Cipher Spec. At this point, the handshake is complete and the client and server may begin to exchange user layer data.

Column 15, line 25, to column 16, line 34:

FIG. 7 is a diagram which illustrates a security module design having clean separation from the browser specific implementations. The security module includes the main COSecurity class 402, and the interface COBrowserSecurityInterface 404. The COSecurity object checks browser type upon instantiation. It does so by requesting the "java.vendor" system property. In the preferred embodiment of the invention, Microsoft Internet Explorer.TM. is the default browser, but if the browser is Netscape, for

example, the class then instantiates by name the concrete implementation of the Netscape security interface, nmco.security.securityimpls. CONetscape4.sub.-- 0SecurityImpl 406. Otherwise, it instantiates nmco.security.securityimpls. CODefaultSecurityImpl 408.

The COBrowserSecurityInterface 404 mirrors the methods provided by COSecurity 402. Concrete implementations such as CONetscape4.sub.-- 0SecurityImpl 406 for Netscape Communicator and CODefaultSecurityImpl 408 as a default are also provided. Adding a new implementation 410 is as easy as implementing the COBrowserSecurityInterface, and adding in a new hook in COSecurity.

After using "java.vendor" to discover what browser is being used, COSecurity 402 instantiates by name the appropriate concrete implementation. This is done by class loading first, then using Class.newInstance( ) to create a new instance. The newInstance( ) method returns a generic object; in order to use it, it must be cast to the appropriate class. COSecurity 402 casts the instantiated object to COBrowserSecurityInterface 404, rather than to the concrete implementation. COSecurity 402 then makes calls to the COBrowserSecurityInterface "object," which is actually a concrete implementation "in disguise." This is an example of the use of object oriented polymorphism. This design cleanly separates the specific implementations which are browser-specific from the browser-independent COSecurity object.

Each COApp object may either create their own COSecurity object using the public constructors, or retrieve the COSecurity object used by the backplane via COBackPlane.getSecurity( ). In general, the developer of the applications to be run will use the COSecurity object whenever the COApp needs privileged access to any local resource, i.e., access to the local disk, printing, local system properties, and starting external processes. The following represents an example of the code generated when using the security object.

```
// Instantiating CoSecurity objectCOSecurity
security=new CoSecurity( );
// Now access a privileged resource
try {
     String s=
     security.getSystemProperty("user.home");
     System.out.println(s);
}
catch(COSecurityException cose)
{
// take care in case of security exception
}
```

> Referring back to FIG. 10, once the browser type has been confirmed, the logon applet checks for the name/password entry and instantiates a session object in step 292, communicating the name/password pair to the enterprise system. The session object sends a message containing the name/password to the StarOE server 49 for user validation in step 294.
>
> When the user is properly authenticated by the server in step 296, another Web page which launches the backplane object is downloaded in steps 298, 300, 304. This page is referred to as a home page. At the same time, all the remaining application software objects are downloaded in CAB or JAR files as indicated at step 302. If the system of the present invention determines that the backplane and application files have been already downloaded, the steps 300, 302, 304 are not performed. The backplane object is then instantiated in step 306.

As should be apparent by merely reading the above-copied portions of Devine et al., the portions of Devine et al. that have been applied against the second element of independent claim 1 do not disclose the use of "a reception bean", as required by the claim language. In fact, Devine et al. does not even mention the use of a bean at all. At most, Devine et al. discloses object-oriented classes and objects; however, a bean is not identical nor equivalent with a generic object-oriented class or object because a bean is an object that has particular operational properties or characteristics. Since Devine et al. does not disclose the use of beans, Devine et al. cannot be used as an anticipatory reference against the claims. At a minimum, since Devine et al. cannot be used as an anticipatory reference, the Office action should have at least attempted to explain how the teachings of Devine et al. could have hypothetically been modified in an obvious manner to reach the claimed invention, which would require an obviousness rejection rather than an anticipatory rejection.

A common rejection was applied against independent claim 1 and dependent claims 3, 15, and 26. Claims 3, 15, and 26 recite that the request implementation software comprises at least one bean. Since Devine et al. does not disclose the use of beans, Devine et al. cannot be used as an anticipatory reference against claims 3, 15, and 26.

Whereas independent claim 1 is directed to an apparatus, independent claim 13 is directed to a method, and independent claim 24 is directed to a computer program product. A common rejection was applied against claims 13 and 24. Claims 13 and 24 contain additional elements, but the rejection of claims 13 and 24 relies on the same argument as the rejection of claim 1 and points to the same portions of Devine et al.. Applicant's argument with respect to the rejection of claim 1 is applicable against claims 13 and 24.

Independent claim 35 reads as follows:

> An apparatus for implementing a public key infrastructure in a distributed processing system, the apparatus comprising:
> a plurality of beans, the beans communicatively coupled to one another and responsive to events generated by the plurality of beans; and
> at least one of the plurality of beans comprising a pipe bean that propagates an event to another of the plurality of beans.

In order to emphasize that Devine et al. does not disclose the claimed features, Applicant copies hereinbelow the portions of Devine et al. that have been applied by the rejection against claim 35.

Column 18, line 3, to column 19, line 22:

> FIG. 11 is a data flow diagram illustrating data flow among the processing modules of the "network MCI Interact" during logon, entitlement request/response, heartbeat transmissions and logoff procedures. As shown in FIG. 11, the client platform includes the networkMCI Interact user 340 representing a customer, a logon Web page having a logon object for logon processing 342, a home page having the backplane object. The Web server 344, the dispatcher server

346, cookie jar server 352, and StarOE server 348 are typically located at the enterprise site.

As described above, following the SSL handshake, certain cab files, class files and disclaimer requests are downloaded with the logon Web page as shown at 440. At the logon Web page, the customer 340 then enters a userid and password for user authentication as illustrated at 440. The customer also enters disclaimer acknowledgment 440 on the logon page 342. If the entered userid and password are not valid or if there were too many unsuccessful logon transactions, the logon object 342 communicates the appropriate message to the customer 340 as shown at 440. A logon object 342, typically an applet launched in the logon Web page connects to the Web server 344, for communicating a logon request to the system as shown at 442. The logon data, having an encrypted userid and password, is sent to the dispatcher 346 when the connection is established as shown at 444. The dispatcher 346 then decrypts the logon data and sends the data to the StarOE 348 after establishing a connection as shown at 446. The StarOE 348 validates the userid and password and sends the results back to the dispatcher 346 as illustrated at 446 together with the user application entitlements. The dispatcher 346 passes the data results obtained from the StarOE 348 to the Web server 344 as shown at 444, which passes the data back to the logon object 342 as shown at 442. The customer 340 is then notified of the logon results as shown as 440.

When the customer 340 is validated properly, the customer is presented with another Web page, referred to as the home page 350, from which the backplane is typically launched. After the user validation, the backplane generally manages the entire user session until the user logs off the "networkMCI Interact." As shown at 448, the backplane initiates a session heartbeat which is used to detect and keep the communications alive between the client platform and the enterprise Intranet site. The backplane also instantiates a COUser object for housekeeping of all client information as received from the StarOE 348. For example, to determine which applications a current customer is entitled to access and to activate only those application options on the home page for enabling the customer to select, the backplane sends a "get application list" message via the Web server 344 and the dispatcher 346 to the StarOE 348 as shown at 448, 444, and 446. The entitlement list for the customer is then sent from the StarOE 348 back to the dispatcher 346, to the Web server 344 and to the backplane at the home page 350 via the path shown at 446, 444, and 448. The application entitlements for the customer are kept in the COUser object for appropriate use by the backplane and for subsequent retrieval by the client applications.

The entitlement information for COUser is stored in a cookie jar 352, maintained in the cookie jar server 32 or the dispatcher server 26 (illustrated in FIGS. 4 and 5). When the Web server receives the entitlement requests from the backplane at the home page 350 or from any other client applications, the Web server 344 makes a connection to the cookie jar 352 and checks if the requested information is included in the cookie jar 352 as shown at 450. The cookie jar 352 is a repository for current customer sessions and the individual session details are included in a cookie including the entitlement information from the OE server 348. During the logon process described above, the OE server 348 may include in its response, the entitlements for the validated customer. The dispatcher 346 transfers the entitlement data to the Web server 344, which translates it into a binary format. The Web server 344 then transmits the binary entitlement data to the cookie jar 352 for storage and retrieval for the duration of a session. Accordingly, if the requested information can be located in the cookie jar 352, no further request to the StarOE 348 may be made. This mechanism cuts down on the response time in processing the request. Although the same information, for example, customer application entitlements or entitlements for corp ids, may be stored in the COUser object and maintained at the client platform as described above, a second check is usually made with the cookie jar 352 via the Web server 344 in order to insure against a corrupted or tampered COUser object's information. Thus, entitlements are typically checked in two places: the client platform 10 via COUser object and the Web server 344 via the cookie jar 352.

Column 24, line 26, to column 25, line 67:

FIG. 13(a) and 13(b) are schematic illustrations showing the message format passed between the dispatcher 26 and the relevant application specific proxy, (FIG. 13(a)) and the message format passed between the application specific proxy back to the Dispatcher 26 (FIG. 13(b)). As shown in FIG. 13(a), all messages between the Dispatcher and the Proxies, in both directions, begin with a common header 150 to allow leverage of common code for processing the messages. A first portion of the header includes the protocol version 165 which may comprise a byte of data for identifying version control for the protocol, i.e., the message format itself, and is intended to prevent undesired mismatches in versions of the dispatcher and proxies. The next portion includes the message length 170 which, preferably, is a 32-bit integer providing the total length of the message including all headers. Next is the echo/ping flag portion 172 that is intended to support a connectivity

test for the dispatcher-proxy connection. For example, when this flag is non-zero, the proxy immediately replies with an echo of the supplied header. There should be no attempt to connect to processes outside the proxy, e.g. the back-end application services. The next portion indicates the Session key 175 which is the unique session key or "cookie" provided by the Web browser and used to uniquely identify the session at the browser. As described above, since the communications middleware is capable of supporting several types of transport mechanisms, the next portion of the common protocol header indicates the message type/mechanism 180 which may be one of four values indicating one of the following four message mechanisms and types: 1) Synchronous transaction, e.g., a binary 0; 2) Asynchronous request, e.g., a binary 1; 3) Asynchronous poll/reply, e.g., a binary 2; 4) bulk transfer, e.g., a binary 3.

Additionally, the common protocol header section includes an indication of dispatcher-assigned serial number 185 that is unique across all dispatcher processes and needs to be coordinated across processes (like the Web cookie (see above)), and, further, is used to allow for failover and process migration and enable multiplexing control between the proxies and dispatcher, if desired. A field 140 indicates the status is unused in the request header but is used in the response header to indicate the success or failure of the requested transaction. More complete error data will be included in the specific error message returned. The status field 140 is included to maintain consistency between requests and replies. As shown in FIG. 13(a), the proxy specific messages 178 are the metadata message requests from the report requester client and can be transmitted via synchronous, asynchronous or bulk transfer mechanisms. Likewise, the proxy specific responses are metadata response messages 180 again, capable of being transmitted via a synchronous, asynchronous or bulk transfer transport mechanism.

It should be understood that the application server proxies can either reside on the dispatcher server 26 itself, or, preferably, can be resident on the middle-tier application servers 40, i.e., the dispatcher front end code can locate proxies resident on other servers.

As mentioned, the proxy validation process includes parsing incoming requests, analyzing them, and confirming that they may include validly formatted messages for the service with acceptable parameters. If necessary, the message is translated into an underlying message or networking protocol. If no errors are found, the proxy then manages the communication with the middle-tier server to actually get the request serviced. The application proxy supports application specific translation and communication

with the back-end application server for both the Web Server (java applet originated) messages and application server messages.

Particularly, in performing the verification, translation and communication functions, the Report Manager server, the Report Scheduler server and Inbox server proxies each employ front end proxy C++ objects and components. For instance, a utils.c program and a C++ components library, is provided for implementing general functions/objects. Various C++ parser objects are invoked which are part of an object class used as a repository for the RM metadata and parses the string it receives. The class has a build member function which reads the string which includes the data to store. After a message is received, the parser object is created in the RMDispatcher.c object which is a file which includes the business logic for handling metadata messages at the back-end. It uses the services of an RMParser class. Upon determining that the client has sent a valid message, the appropriate member function is invoked to service the request. Invocation occurs in MCIRMServerSocket.C when an incoming message is received and is determined not to be a talarian message. RMSErverSocket.c is a class implementing the message management feature in the Report Manager server. Public inheritance is from MCIServerSocket in order to create a specific instance of this object. This object is created in the main loop and is called when a message needs to be sent and received; a Socket.c class implementing client type sockets under Unix using, e.g., TCP/IP or TCP/UDP. Socket.C is inherited by ClientSocket.C:: Socket(theSocketType, thePortNum) and ServerSocket.C:: Socket(theSocketType, thePortNum) when ClientSocket or ServerSocket is created. A ServerSocket.c class implements client type sockets under Unix using either TCP/IP or TCP/UDP. ServerSocket.C is inherited by RMServerSocket when RMServerSocket is created. An InboxParser.c class used as a repository for the RM Metadata. The class' "build"member function reads the string which includes the data to store and the class parses the string it receives. After a message has been received, the MCIInboxParser object is created in inboxutl.c which is a file which includes the functions which process the Inbox requests, i.e, Add, Delete, List, Fetch and Update.

As should be apparent by merely reading the above-copied portions of Devine et al., the portions of Devine et al. that have been applied against independent claim 35 do not disclose the use of "beans" or "a pipe bean", as required by the claim language. In fact, Devine et al. does not even mention the use

of a bean at all. At most, Devine et al. discloses object-oriented classes and objects; however, a bean is not identical nor equivalent with a generic object-oriented class or object because a bean is an object that has particular properties or characteristics. Since Devine et al. does not disclose the use of beans, Devine et al. cannot be used as an anticipatory reference against the claims.

A common rejection was applied against independent claim 35 and dependent claims 4, 16, 27, and 35. Claims 4, 16, and 27 recite that an apparatus, method, or computer program product comprises at least a pipe bean. Since Devine et al. does not disclose the use of beans, Devine et al. cannot be used as an anticipatory reference against claims 4, 16, and 27.

Whereas independent claim 35 is directed to an apparatus that includes a pipe bean, independent claim 44 is directed to an apparatus that includes beans without specifically reciting a pipe bean. However, the rejection of claim 44 pointed to the same portions of Devine et al. as were used against claim 35. Applicant's argument with respect to the rejection of claim 35 is applicable for claim 44.

With respect to the remaining dependent claims, all of the rejections point to the same portions of Devine et al. that were applied against the claims that have been discussed hereinabove. Since Devine et al. does not disclose the use of beans, Devine et al. cannot be used as an anticipatory reference against these claims.

Devine et al. clearly does not disclose features as required by the language of the claims of the present application. As stated at MPEP § 2131: "A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). "The

identical invention must be shown in as complete detail as is contained in the ... claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). Hence, for this and other reasons, Devine et al. cannot be used as an anticipatory reference, and the rejections of the claims have been overcome, whereby Applicant requests the withdrawal of the rejections.

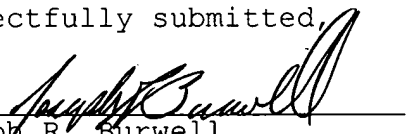
VI. Conclusion

It is respectfully urged that the present patent application is patentable, and Applicant kindly requests a Notice of Allowance.

For any other outstanding matters or issues, the examiner is urged to call or fax the below-listed telephone numbers to expedite the prosecution and examination of this application.

DATE: January 3, 2005

Respectfully submitted,



Joseph R. Burwell
Reg. No. 44,468
ATTORNEY FOR APPLICANT

Law Office of Joseph R. Burwell
P.O. Box 28022
Austin, Texas 78755-8022
Voice: 866-728-3688 (866-PATENT8)
Fax: 866-728-3680 (866-PATENT0)
Email: joe@burwell.biz